ERDC MSRC/PET TR/00-08

Dual-Level Parallelism Improves Load-Balance in Coastal Ocean Circulation Modeling

by

Phu Luong
Clay P. Breshears
Le N. Ly

10 March 2000

03h0032000

Work funded by the DoD High Performance Computing Modernization Program CEWES Major Shared Resource Center through

Programming Environment and Training (PET)

Supported by Contract Number: DAHC94-96-C0002
Nichols Research Corporation

Dual-Level Parallelism Improves Load-Balance in Coastal Ocean Circulation Modeling

Phu Luong*
Center for Subsurface Modeling
University of Texas, Austin
ERDC MSRC, CEERD-IH-C
3909 Halls Ferry Road
Vicksburg, MS 39180
phu@nrcmail.wes.hpc.mil
fax: 601-634-3808

Clay P. Breshears†
Center for High Performance Software Research
Rice University
ERDC MSRC, CEERD-IH-C
3909 Halls Ferry Road
Vicksburg, MS 39180
clay@turing.wes.hpc.mil
fax: 601-634-3808

Le N. Ly
Department of Oceanography
Naval Postgraduate School
833 Dyer Road
Monterey, CA 93943
lely@hydro.oc.nps.navy.mil
fax:831-656-2712

March 2, 2000

*On-Site EQM Lead, ERDC MSRC
†On-Site SPP Tools Lead, ERDC MSRC

Dual-Level Parallelism Improves Load-Balance in Coastal Ocean Circulation Modeling

Abstract

Numerical grid generation techniques play an important role in the numerical solution of partial differential equations on arbitrarily-shaped regions. For coastal ocean modeling, in particular, a one-block grid covering the region of interest is most commonly used. Most bodies of water have complicated coastlines; e.g., Persian Gulf and Mediterranean Sea. In such a physical domain, the number of unused grid points can be a relatively large portion of the entire domain space. Other disadvantages of using a one-block grid include large memory requirements and poor resolution for a large body of water; e.g., Pacific Ocean.

In this study, we introduce a multi-block grid generation technique and a dual-level parallel implementation to eliminate these problems. Message Passing Interface (MPI) is used to parallelize the Princeton Ocean Model (POM) ocean circulation code such that each grid block is assigned to a unique processor. Since not all grid blocks are of the same size, the work-load varies between MPI processes. To alleviate this we use dynamic threading to improve load balance. Performance results from the `POM` model on both a one-block grid and twenty-block grid after a 90-day simulation for the Persian Gulf demonstrate the efficacy of the MPI-only and MPI/OpenMP code versions.

Keywords: Princeton Ocean Model, dual-level parallelism, multi-block grid, Persian Gulf simulation, coastal ocean circulation model.

1 Introduction

Poor computational grid resolution for large bodies of water has long been a problem for ocean modeling. Sometimes, coarse grid resolution is not sufficient for planning military operations. To overcome this difficulty, many modelers use nesting techniques, in which a fine grid resolution is used in the operation area (nested grid) and results from the ocean model on a coarse grid (hosted grid) are used as boundary conditions for the nested grid. Nesting techniques, however, generate nonphysical reflecting waves across the interface between the hosted grid and the nested grid [1].

Over the years, the traditional one-block rectangular grid has been used for ocean circulation modeling. This technology encounters difficulty on computational grids with high resolution due to the large memory and processing requirements. For a large body of water, such as an ocean, with complicated coastlines, the number of grid points used in the calculation (water points) is often the same or even smaller than the number of unused grid points (land points). It is known that domain decomposition can be used to partition the traditional one-block grid into sub-domains that reduce the unused grid points and improve performance of the ocean model [2]. MPI [3] can be used to parallelize this type of computation. A MPI implementation with domain decomposition often requires a preprocessing step to determine the most efficient work distribution for the sub-domains in order to avoid severe load imbalances [2]. Load imbalance adversely affects parallel performance and scalability.

We propose a multi-block grid generation technique and parallel implementation of the Princeton Ocean Model (POM) ocean circulation code [4]. The multi-block

grid generation technique allows us to eliminate blocks composed mainly of land grid points. It also allows us to choose the grid with minimum land points along the coastline. In addition, a high horizontal resolution of an area of interest can be handled easily. The advantages of multi-block grid generation techniques over the traditional rectangular one-block grid and nesting techniques can be found in [5].

We describe the use of MPI and OpenMP [6] to exploit two levels of parallelism in the multi-block computation. Since not all grid blocks are of the same size, we use the OpenMP dynamic threading feature to improve the performance and load balance in the POM code [7]. We have chosen the modeling of the Persian Gulf as our prototypical problem to demonstrate the effectiveness of the multi-block grid technique utilizing the dual-level parallel execution model.

We first describe our method for creating a multi-block grid from a single block grid in Section 2. In Section 3, a review of the POM code and the physics modelled, as well as the numerical simulation, is presented. Details of the parallel implementations (MPI-only and MPI/OpenMP) are given in Section 4. Performance results of the parallel codes are included in this section as well. Our conclusions from this study are presented in Section 5.

2 Multi-Block Grid Generation

A one-block rectangular grid and twenty-block grid are both used for this study. The one-block grid was generated by a simple algebraic scheme using the EAGLEView software package [8]. EAGLEView is an interactive surface and volumetric

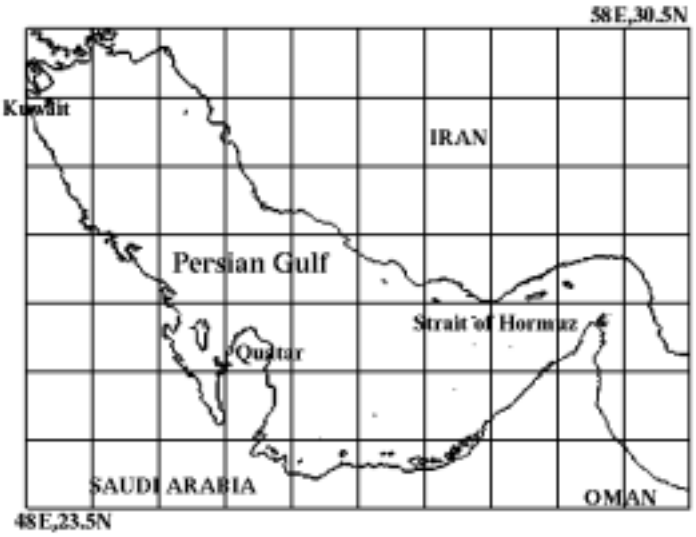
grid generation program developed by the Engineering Research Center at Mississippi State University. The twenty-block grid is a decomposition of the one-block rectangular grid with the same resolution. The decomposition is done through the use of several modules in EAGLEView. The I/O module allows the user to load in the coastline data and the one-block grid data. The extract module is then able to extract the land grid portion from the domain. Through the EAGLEView GUI, this same module is used for decomposing the water grid portion into small blocks. After having the water portion of the domain decomposed into blocks, the I/O module is used again for writing coordinates values of each grid block into separate files.

2.1 Persian Gulf

The physical geographic area we chose for this study is the Persian Gulf. This area extends from 48 East to 58 East in longitude and from 23.5 North to 30.5 North in latitude. Part of the Gulf of Oman is also included in this physical domain. Figure 1 shows the coastlines and geographic information for the study region. The complicated features of the coastlines near Qatar, along the Strait of Hormuz, and the northern part of the Persian Gulf present an opportunity to demonstrate the advantages of multi-block grids within the POM code.

Figure 2 shows the one-block computational grid of the Persian Gulf region with dimensions 291x211 (61,401 total grid points). The number of relevant grid points in the one-block grid is 22,309, while the number of unused grid points is 39,092. However, the twenty-block grid (Figure 3) contains a total of 32,031 grid points

Figure 1 The Persian Gulf Coastline



with only 9,722 of those as unused. Approximately 70% of the total grid points are actually used in the twenty-block grid calculation compared to only 36% in the one-block rectangular grid. Also, since the twenty-block grid contains almost half the total number of points as the one-block grid, we should expect to save up to 50% of the required memory needed to hold the grid model. We would also expect a significant reduction in serial execution time.

3 Model Descriptions and Simulations

The coastal ocean is a region receiving a great deal of attention due to an increased utilization for human habitation, aquatic development and military operations. These activities require a knowledge of dynamic and thermodynamic struc-

Figure 2 One-block computational grid with used and unused grid points

Figure 3 Twenty-block computational grid with used and unused grid points

tures of the coastal regions such as water circulation, ocean wave dynamics, storm surges, and evolution of seawater temperature and salinity. A review of the POM three-dimensional, primitive equation, time-dependent, σ coordinate, free surface coastal ocean circulation model is presented in this section.

3.1 Governing Equations

The model primitvie equations describe the velocity, surface elevation, salinity, and temperature fields in the ocean. The ocean is assumed to be hydrostatic and incompressible (Boussinesq approximation).

The equations are written in a Cartesian coordinate system with x eastward, y northward, and z upward. The free surface is located at $z = \eta(x, y, t)$ and the bottom is at $z = -H(x, y)$.

The governing equations used in the POM model are:

$$\frac{\partial \vec{V}}{\partial t} + \vec{V} \cdot \nabla \vec{V} + W \frac{\partial \vec{V}}{\partial z} + 2\vec{\Omega} \times \vec{V} = -\frac{1}{\rho_0} \nabla P + \frac{\partial}{\partial z}(K_m \frac{\partial \vec{V}}{\partial z}) + \vec{F} \tag{1}$$

$$\frac{\partial P}{\partial z} = -\rho g \tag{2}$$

$$\nabla \cdot \vec{V} + \frac{\partial W}{\partial z} = 0 \tag{3}$$

$$\frac{d\theta_i}{\partial t} + \vec{V} \cdot \nabla \theta_i + W \frac{\partial \theta_i}{\partial z} = \frac{\partial}{\partial z}(K_h \frac{\partial \theta_i}{\partial z}) + F_{\theta_i}. \tag{4}$$

The density is computed by using the equation of state in the general form:

$$\rho = \rho(\theta, S, p). \tag{5}$$

The balance of momentum is described by Equation (1), Equation (2) is the hydrostatic equation, Equation (3) is the continuity equation and the conservation equations for temperature and salinity are described in Equation (4). The Coriolis force is denoted as $2\vec{\Omega} \times \vec{V}$, where $\vec{\Omega}$ is the earth rotation vector, $\vec{V}$ is the horizontal velocity vector with components (U, V), ∇ is the horizontal gradient operator, ρ_o is the reference density, ρ is the seawater density, g is the gravitational acceleration, P is the pressure, and K_m and K_h are the vertical turbulent exchange coefficients for momentum of heat and salt, respectively. In Equation (4), θ_i may represent mean potential temperature, θ, or salinity, S. The horizontal diffusion terms $\vec{F}(F_x, F_y)$ in Equation (1) and F_{θ_i} in Equation (4) can be calculated using Smagorinsky horizontal diffusion formulation [9].

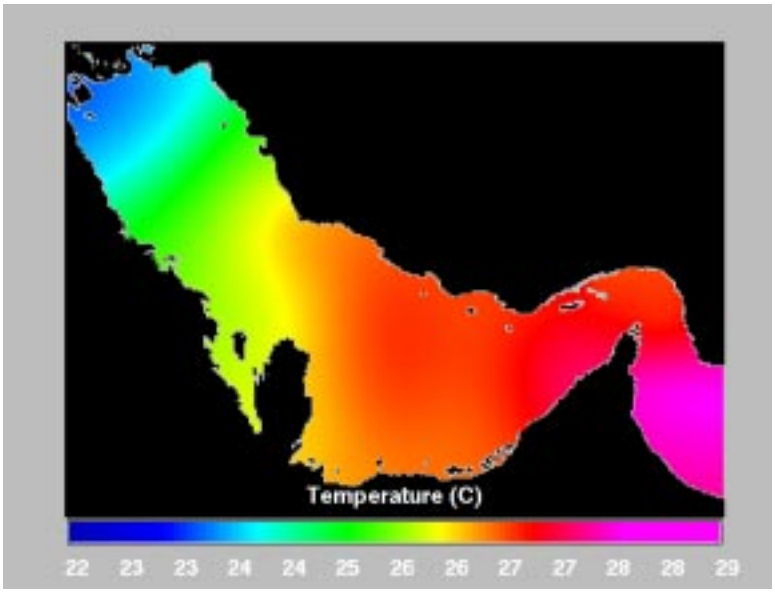
The diffusion terms in Equations (1) and (4) contain the vertical turbulent exchange coefficients which are determined by the second order turbulence closure scheme of Mellor and Yamada [10]. The turbulence scheme is characterized by equations for turbulent kinetic energy (TKE), $q^2/2$, and for the turbulent mixing length, ℓ. The equations can be written in the same form for function Q_i, so that Q_i is either $q^2/2$ for TKE or $q^2\ell$ for the turbulent mixing length [11]. Detail of the turbulence closure used in POM ocean circualtion model can be found in [10] as well.

3.2 Persian Gulf Simulation

The Persian Gulf is a shallow embayment of the Gulf of Oman, and the average depth of the basin is only about 35 m. The greatest depths in the Persian Gulf are approximately 150 m, and are located in the central basin and at the Strait of Hormuz. The bathymetry data for both the one-block grid and twenty-block grid are obtained from the NAVOCEANO's two minutes resolution bathymetry database by interpolation. Figure 5 shows a color contour of bathymetry data on the computational grid model. The portion of the Gulf of Oman used in this data set, where the open boundary condition is imposed ranges to a depth of about 2000 m.

The POM model has realistic coastlines and bottom topography with 26 levels of bottom-following vertical σ coordinate. The model is initialized with the NAVOCEANO's annual ten minutes resolution temperature and salinity Generalized Digital Environmental Model (GDEM) database. Figure 4 shows the surface temperature on the computational grid obtained by interpolation from the GDEM database. At the open boundary the internal normal velocities are governed by a Sommerfeld radiation condition. The open boundary condition for the surface elevation is zero gradient normal to the boundary. Temperature, salinity and tangential velocities are upwinded at the open boundary. The model is spun up for 30 days (diagnostic mode) in which the density distribution at all points on the computational grids are held fixed in time.

Figure 4 Surface temperature on one-block grid

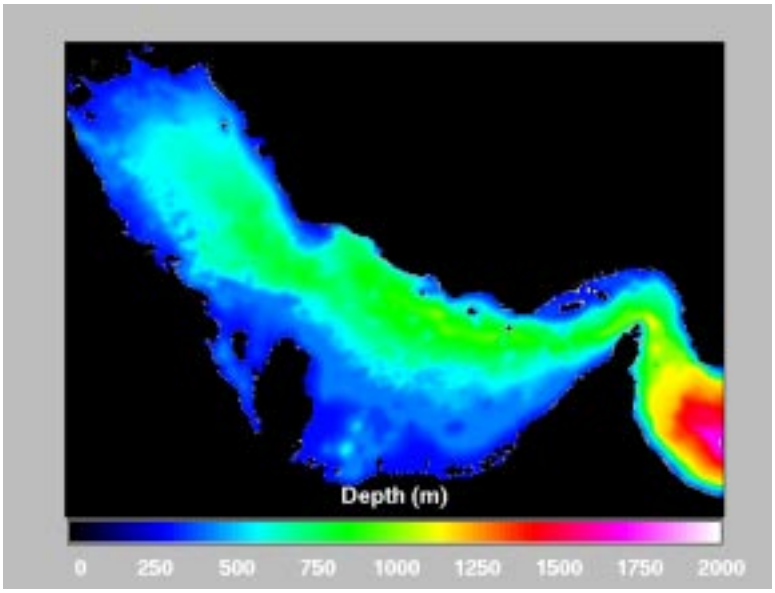


3.3 Multi-Block Grid Serial Performance

Throughout the multi-block grid serial computation, each grid block is considered to have four interfaces (west, south, east, north) for exchange of overlapping grid points with other blocks. Information is updated after every time step in sequential order from block number one to block number twenty. The time step is 40 s for the external mode and 240 s for the internal mode. After 30 days of diagnostic mode the model is then run for 60 days. Numerical solutions after a 90-day simultion of the serial code version on the one-block grid and twenty-block grid yield identical results.

Model output of the Persian Gulf annual surface circulation for temperature on the one-block grid and twenty-block grid is shown in Figures 6 and 7, respectively. The simulations were computed on the SGI Origin2000 at U. S. Army Engineer

Figure 5 Bathymetry on one-block grid



Research and Development Center (ERDC) Major Shared Resource Center (MSRC) in Vicksburg, MS. Serial execution time for a 10-day simulation on the one-block grid was over 21 hours as compared to just over 16 hours for the twenty-block grid. These results show the twenty-block grid serial performance was not overly encouraging, however, we have saved 50% of the memory requirement of the one-block grid code.

4 Parallelization of Multi-Block POM

POM is a standard Fortran code that was initially designed for serial computers and later ported to vector machines. In the POM multi-block grid version, each rectangular block grid is considered to have four neighboring interfaces (west, south,

Figure 6 Surface temperature on one-block grid after a 90-day simulation

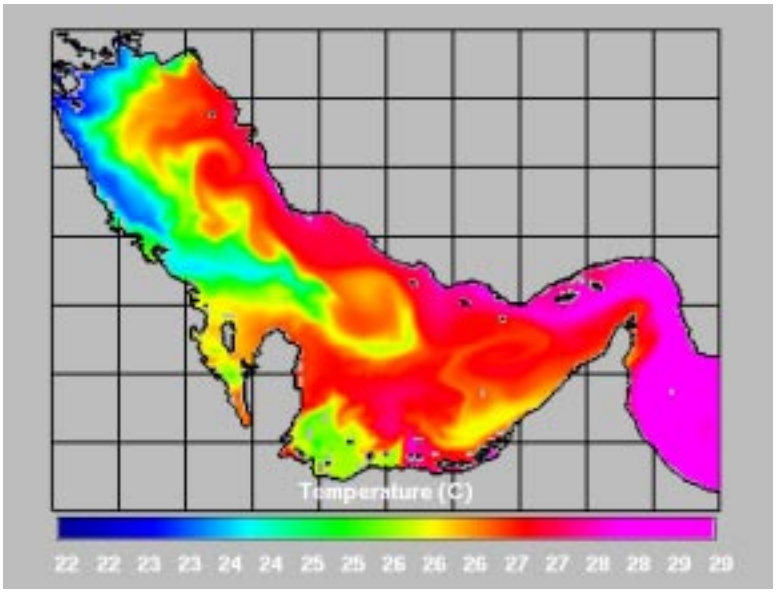
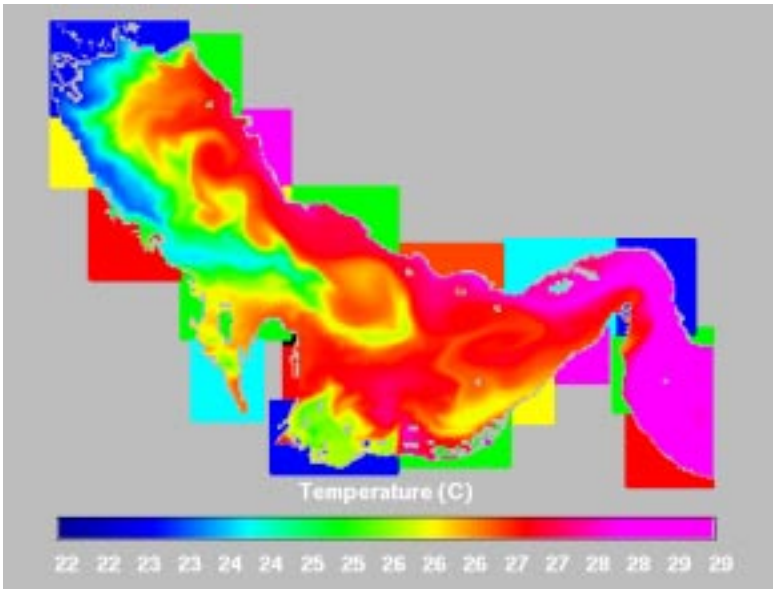


Figure 7 Surface temperature on twenty-block grid after a 90-day simulation



east, and north) with adjacent blocks. Special considerations for computing with interfaces bordering open ocean, or those without adjacent blocks, are built into the model. While there can be only one shared interface between two adjacent blocks, a block may have any number of adjacent blocks along a given block face. For simplicitiy, the code used in this study restricted the number of different blocks adjacent to any one face at no more than two. (We anticipate no problems with allowing an unknown number of adjacent blocks per face and are planning to modify the code to handle such data sets in a future version.)

4.1 MPI

The serial version of the code includes four routines specifically designed to transfer data between grid blocks. Processing blocks from the first to the last in order, data is copied from those blocks that are adjacent to the current block and stored within the appropriate ghost cells. After all blocks have been processed, the result is an exchange of data between adjacent blocks; i.e., the amount of data copied from one block into another is the same size as the data copied in the other direction.

With message passing these updates can be done in parallel. Because of the symbiotic nature of the data exchanges, the amount of data that each block must receive from each adjacent block must also be sent to those adjacent blocks. In the serial code, the order of updates was governed by processing all adjacent blocks on one interface before proceeding to the next interface (chosen by moving around the compass points of the interfaces).

Data is sent to all adjacent blocks within the same interface and the interfaces

are processed in order as in the serial version. However, asynchronous receives are posted by each block after each set of data is sent. After all data has been sent, each block processes the actual receipt of data. The exchange between blocks is completed after the data has arrived in the block's processor and been moved into the appropriate overlapping grid points. This use of asynchronous sends has three advantages:

1. Hides message latency since blocks may continue to send data to other blocks without regard to whether or not corresponding exchange data has been received.

2. Avoids deadlock by ensuring that all blocks are not waiting for a receipt of data from an adjacent block that is expecting to receive data from the same compass point interface.

3. Reduces coding complexity due to the fact that attempting to perform synchronous exchanges of data would require adding code to choose and coordinate certain blocks to exchange along their western interface while others exchange along their eastern interface.

The order of synchronous exchanges between blocks with two or more adjacent blocks along an interface would need to be coordinated correctly. Failure to perform the exchanges in the proper order would result in, at best, a serialization of the exchange process or, at at worst, a deadlock.

Since all blocks synchronize to some extent at the communication routines, those blocks with less computation to be done will tend to spend more time waiting for

completion of data exchanges than blocks with more grid points. Normally this would be a cause for concern in many parallel applications and would likely point out that the load balance among MPI tasks needed to be balanced more closely. However, in order to balance the workload more evenly in the multi-block version of POM would require a complete restructuring of the grid blocks in the data set. Another means of creating a more balanced execution time between updates would be to further parallelize the computations in those blocks that have been assigned larger numbers of grid points. We explore methods to achieve this in the next section.

4.2 OpenMP

OpenMP is a collection of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism. OpenMP allows the POM code to be parallelized at a fine granularity (Fortran DO loop level).

Profiling the serial code revealed several routines that accounted for more than half of the total execution time. The VAMPIR (Pallas) performance analysis tool was used to identify individual loops that resulted in the majority of the execution time within these selected routines. A number of these loops were chosen and each was analyzed for data dependencies before OpenMP directives were inserted. Results from these initial experiments were not overly encouraging. A slight decrease in the overall execution time was demonstrated with four OpenMP threads per process.

Many different loops within the code were prime candidates for OpenMP di-

rectives. However, we were daunted by the task of manually inserting directives throughout the code. Fortunately, the MIPSpro version 7.30 Fortran 90 compiler on the SGI Origin2000 is equipped with an Auto-Parallelizing Option flag (`-apo`) to automatically analyze loop dependencies and insert OpenMP directives where it is safe to do so. From previous experience with the dynamic threading feature of OpenMP [7] and the similar load imbalance characteristics inherent in the multi-block grid structure, we examined two methods of using OpenMP with a dynamic number of threads.

Our first method involved manual control of the number of threads spawned by each process in conjunction with compiling selected routines with the `-apo` flag. A threshold of the minimum amount of work needed to spawn a thread is set and each process computes the number of threads (up to some set maximum number) that should be used based on the assigned workload. The OpenMP routine OMP_SET_NUM_THREADS is called after the number of threads for the process has been found. When a process determines that a single thread is to be used, the original serial routine is called, otherwise the version with added directives is called. It is assumed that the overhead needed to create a single OpenMP thread, often multiple times within a single routine, is avoided. We refer to this method as manual MPI/OpenMP.

The second method was to use the Auto-Parallelizing Option flag when compiling all routines in the POM code. Before execution the environment variable OMP_SET_DYNAMIC is set to .TRUE.; this allows the SGI Origin2000 runtime system to choose the number of threads spawned by a process at each OpenMP.

One potential drawback to this method is the overhead of spawning a single thread to process a small data set. We refer to this method as automatic MPI/OpenMP

Obviously, the first version of dynamic threading requires much more work and planning than the latter method. However, the first method gives a better improvement in the execution load balance within POM routines. Results from three different executions (MPI-Only, manual MPI/OpenMP, and automatic MPI/OpenMP) are discussed in the next section.

4.3 Parallel Performance

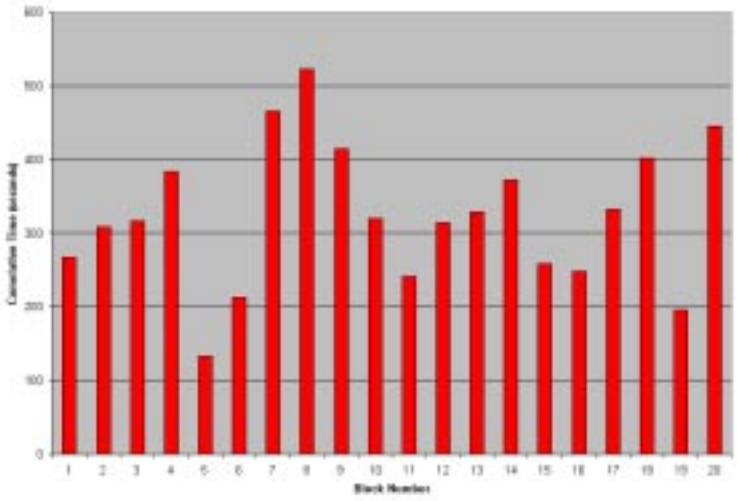
Synchronous communication occurs periodically between the MPI processes during the computation. Therefore, MPI processes with less work (i.e., smaller domains) must wait for slower processes to complete before proceeding with the computation. In order to quantify the degree of load imbalance within a given code segment, we define *idle time* as the ratio of execution time to maximum execution time expressed as a percentage:

$$\%\text{idle} = 100\% - \frac{\sum_{i=0}^{N-1} t_i}{N \times t_{max}} \tag{6}$$

where N is the number of MPI processes, t_i is execution time of process i, and t_{max} is the largest time t_i.

Profiling the serial POM code with the SpeedShop profiling tool on the SGI Origin2000 revealed several routines which consumed over half of the execution time. The top one among those routines is `PROFQ` which takes nearly 20% of the total execution time in each processor.

Figure 8 PROFQ (MPI-only) Cumulative Execution Time in seconds



All results presented herein are for the twenty-block grid data set. Figure 8 shows the timing results of the `PROFQ` routine of the MPI-only code for a run of ten simulated days running on 20 SGI Origin2000 processors. A measure of 38% idle time was committed within this routine. The total wall-clock execution time of this POM code run was 108 minutes.

With the manual MPI/OpenMP version of the code run on 20 processors with a maximum of four OpenMP threads per process, a measure of 21.1% idle time was committed. Figure 9 shows the `PROFQ` routine timing results for this version of the code. Under the automatic MPI/OpenMP code, a measure of 29% idle time was achieved within the `PROFQ` routine (Figure 10). While the manual MPI/OpenMP code had the best load balance of the three code versions, it performed worse in the overall wall-clock execution time (46 minutes) than did the automatic MPI/OpenMP code (32 minutes).

Figure 9 PROFQ (manual MPI/OpenMP) Cumulative Execution Time in seconds

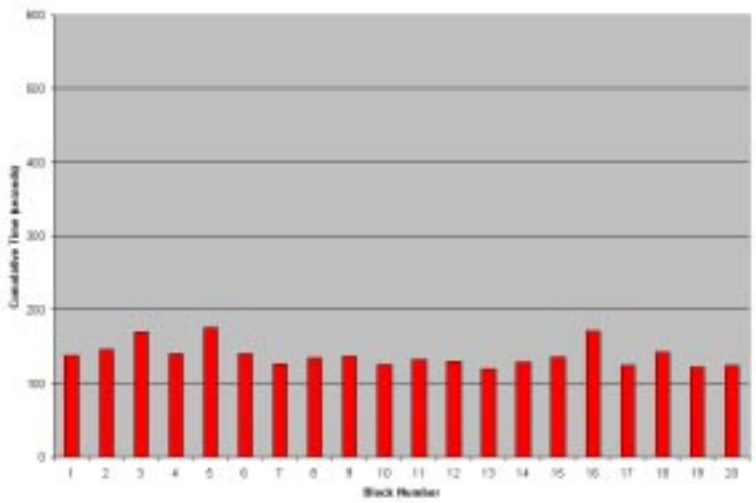
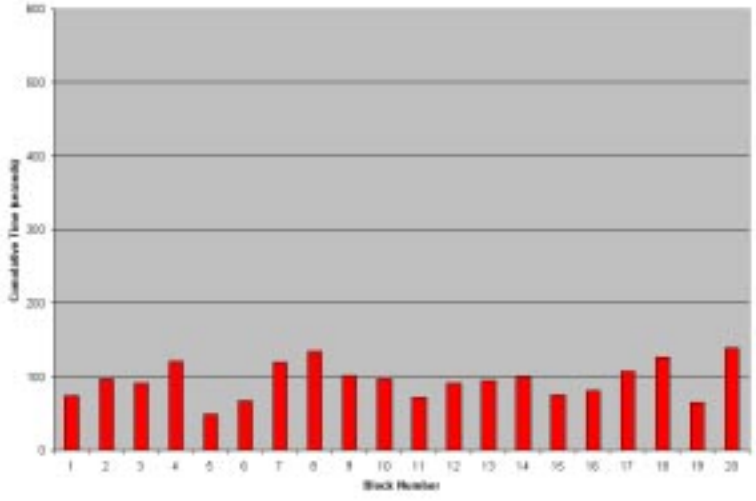


Figure 10 PROFQ (automatic MPI/OpenMP) Cumulative Execution Time in seconds



5 Conclusions

We have demonstrated that the parallel multi-block grid techniques applied to the POM ocean circulation model will eliminate those problems inherent in one-block structured grid codes The timing results show a significant improvement in the execution time as well as in the load imbalance produced by MPI-only execution. Numerical solutions after a 90-day simulation of the model on the one-block and twenty-block yield identical results.

We have also shown that through use of the dynamic threading feature within OpenMP, load balance between the MPI processes can be improved. By using the dual-level algorithm and the grid generation technique presented in this study, we were able to run a 10-day simulation of the Persian Gulf in less than one-half hour as compared to 21 hours for the traditional serial one-block grid version of the code. We were able to achieve a twenty times speed up for the MPI-only version on 20 SGI Origin2000 processors and a forty-five times speed up for the automatic MPI/OpenMP version.

The serial performance of the multi-block grid was not overly encouraging, however, we were able to save almost 50% of the memory requirement of the one-block grid code. With such a significant improvement in performance and memory reduction, we now can apply these techniques for ocean circulation simulations on a larger ocean data set with a higher horizontal resolution.

Future Work

As mentioned earlier, one advantage of the multi-block grid generation technique over domain decomposition is that the multi-block grid allows us to generate a grid with a minimum number of land points along the coastline. Grids can be packed with a high horizontal resolution in any area where needed. As a follow up to this study we plan a performance study on a forty-block grid with a more even work load distribution. Another study would involve packing high resoltion grids where needed. These studies are intended to show the advantages of the multi-block method over both nesting techniques and parallelization by domain decompostion techniques.

Disclaimer

Acknowledgement

Work funded by the DoD High Performance Computing Modernization Program U. S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center through Programming Environment and Training (PET). Supported by Contract Number: DAHC 94-96-C0002 Computer Sciences Corporation.
Authors would like to acknowledge Dr. Henry Gabb, Director of Scientific Computing at ERDC MSRC, Dr. Wayne Mastin, PET Academic Lead at ERDC MSRC and Dr. Mary Wheeler, Director of Center for Subsurface Modeling at UT Austin, TX for supporting this project. We also would like to acknowledge the support of the Office of Naval Research under the NOMP program (Dr. M. Fiadeiro).

References

[1] Spall, M. A. and Holland, W. R. "A nested primitive equation model for oceanic application," *J. Physic. Oceanogr.*, **21**, 205-220, 1991.

[2] Oberpriller, W. D., Sawdey, A. C., O'Keefe, M. T., and Gao, S., "Paralleling the Princeton Ocean Model Using TOPAZ," `http://topaz.lcse.umn.edu`.

[3] Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J., *MPI—The Complete Reference: Volume 1, the MPI Core*, MIT Press, Cambridge, 1998.

[4] Blumberg, A. F. and Mellor, G. L., "A description of a three-dimensional coastal ocean circulation model." In *Three-Dimensional Coastal Models,* Coastal and Estuaries Sciences. Heaps, N. S. editor, AGU Geophysical Monograph Board, 1987, 1.

[5] Ly, N. L., and Luong, P. V., "Numerical Multi-Block Grids in Coastal Ocean Circulation Modeling," *Journal of Applied Mathematical Modeling*, **23**, pp. 865–879, November 1999.

[6] OpenMP Architecture Review Board, "OpenMP Fortran Application Program Interface, Version 1.0," `http://www.openmp.org`, October, 1997.

[7] Luong, P. V., Breshears, C. P., and Gabb, H. A., "Dual-Level Parallelism Improves Load-Balance in the Production Engineering Application CH3D," a techinal report to ERDC DoD HPC MSRC, 1999.

[8] Stokes, M., Jiang, M., and Remotique, M., "EAGLEview Grid Geration Package," EAGLEView Version 2.4 Manual. Missisippi State University/National Science Foundation, Engineering Research Center for Computational Field Simulation, December 1992.

[9] Smagorinsky, J., "General Circulation Experiments with the Primitive Equations. I. The basic Experiment," *Mon. Weather Rev.*, **91**, 99-164, 1963.

[10] Mellor, G. L. and Yamada, T. "A hierachy of turbulence closure models for planetary boundary layers." *J. Atmos. Sci.*, **31**, 1791-1896, 1982.

[11] Ly, L. N. "The Gulf of Mexico responses to Hurricane Frederic simulated with the Princeton numerical ocean circulation model." Technical report, INO, Stennis Space Center, MS, 42 p., 1992.